Express Mail Label No. EV303487112US

COMPUTER ARCHITECTURE AND SYSTEM FOR EFFICIENT MANAGEMENT OF BI-DIRECTIONAL BUS

CROSS-REFERENCE TO RELATED APPLICATIONS

[0001] This application claims priority to U.S. Application serial number 09/652,323 filed August 31, 2000 and incorporated herein by reference. This application also relates to the following commonly assigned co-pending applications entitled:

"Apparatus And Method For Interfacing A High Speed Scan-Path With Slow-Speed Test Equipment," Serial No. 09/653,642, filed August 31, 2000; "Priority Rules For Reducing Network Message Routing Latency," Serial No. 09/652,322, filed August 31, 2000; "Scalable Directory Based Cache Coherence Protocol," Serial No. 09/652,703, filed August 31, 2000; "Scalable Efficient I/O Port Protocol," Serial No. 09/652,391, filed August 31, 2000; "Efficient Translation Lookaside Buffer Miss Processing In Computer Systems With A Large Range Of Page Sizes," Serial No. 09/652,552, filed August 31, 2000; "Fault Containment And Error Recovery Techniques In A Scalable Multiprocessor," Serial No. 09/651,949, filed August 31, 2000; "Speculative Directory Writes In A Directory Based Cache Coherent Nonuniform Memory Access Protocol," Serial No. 09/652,834, filed August 31, 2000; "Special Encoding Of Known Bad Data," Serial No. 09/652,314, filed August 31, 2000; "Broadcast Invalidate Scheme," Serial No. 09/652,165, filed August 31, 2000; "Mechanism To Track All Open Pages In A DRAM Memory System," Serial No. 09/652,704, filed August 31, 2000; "Programmable DRAM Address Mapping Mechanism," Serial No. 09/653,093, filed August 31, 2000; "An Efficient Address Interleaving With Simultaneous Multiple Locality Options," Serial No. 09/652,452, filed August 31, 2000; "A High

Performance Way Allocation Strategy For A Multi-Way Associative Cache System," Serial No. 09/653,092, filed August 31, 2000; "Method And System For Absorbing Defects In High Performance Microprocessor With A Large N-Way Set Associative Cache," Serial No. 09/651,948, filed August 31, 2000; "A Method For Reducing Directory Writes And Latency In A High Performance, Directory-Based, Coherency Protocol," Serial No. 09/652,324, filed August 31, 2000; "Mechanism To Reorder Memory Read And Write Transactions For Reduced Latency And Increased Bandwidth," Serial No. 09/653,094, filed August 31, 2000; "System For Minimizing Memory Bank Conflicts In A Computer System," Serial No. 09/652,325, filed August 31, 2000; "Computer Resource Management And Allocation System," Serial No. 09/651,945, filed August 31, 2000; "Input Data Recovery Scheme," Serial No. 09/653,643, filed August 31, 2000; "Fast Lane Prefetching," Serial No. 09/652,451, filed August 31, 2000; "Mechanism For Synchronizing Multiple Skewed Source-Synchronous Data Channels With Automatic Initialization Feature," Serial No. 09/652,480, filed August 31, 2000; "Mechanism To Control The Allocation Of An N-Source Shared Buffer," Serial No. 09/651,924, filed August 31, 2000; and "Chaining Directory Reads And Writes To Reduce DRAM Bandwidth In A Directory Based CC-NUMA Protocol," Serial No. 09/652,315, filed August 31, 2000, all of which are incorporated by reference herein.

BACKGROUND

[0002] Computer systems typically incorporate one or more buses to facilitate communication between devices or components in the system. As used herein, a bus is a plurality of wires or conductors to which multiple agents or devices are coupled in order to transport data or signals among and between the agents or devices. A bi-directional bus provides for reads and writes (*i.e.*, communications in both directions) to occur on common wires or conductors. Typically, a bus has a certain protocol which is to be followed by all of the agents coupled to the bus. Having a consistent protocol ensures all agents use the same rules of communication on the bus.

[0003] Since a bus is essentially a group of shared conductors or wires, it is important that the agents share the bus in the manner prescribed by the bus

protocol. Moreover, it is important that only one agent drive the bus (*i.e.*, issue or place signals on the shared wires of the bus) at a time. When multiple agents attempt to drive the bus at the same time, it is called a bus conflict or bus contention. Bus contention will often result in the signals or data on the bus being corrupted or unreliable and may also result in damage to the agents on the bus.

[0004] To avoid bus contention, dead time or "bubbles" may be introduced on the bus between bus transactions. Bubbles ensure the last transaction is complete before the next transaction is attempted. The use and necessity of bubbles is dictated by the bus protocol which may in turn be dictated by the agents or devices connected to the bus. In some bi-directional buses, for example the RAMbus® a standard memory bus, a bubble is only required for transitions from reads to writes or vice versa. In computer systems employing such bi-directional buses, then, the system must incorporate some mechanism to insert bubbles between read/write transitions to ensure that multiple agents connected to the bus do not attempt to simultaneously drive the bus, *i.e.*, and to ensure there is no bus contention. This bubble or delay time between read/write transitions ensures that the previous transaction, either a read or a write, has ended before the next transaction is attempted. Note that for buses of this type, when the same kind of transaction occurs consecutively on the bus (*i.e.*, consecutive reads or consecutive writes), a delay is not required to ensure there will be no bus contention and thus no bubble is added. Bubbles are only introduced when the bus transactions switch from a read to a write or vice versa.

[0005] Although bubbles may be necessary to ensure no bus contention, their occurrence should be minimized because the bubbles result in unused bandwidth that would otherwise be useful to the system for enhanced performance. Specifically, the more bubbles that are introduced, the more wait or delay time that is introduced into the system. Accordingly, a system designers desire to maximize bandwidth and system performance is often at odds with the need to ensure avoidance of bus conflicts by adding bubbles.

[0006] One method of reducing the number of bubbles, and the associated delays, would be to stream or group reads and writes together whenever possible. Streaming reads and writes consecutively reduces the number of

transitions from reads to writes or vice versa, thereby reducing the number of bubbles required. The system used to group like transactions, however, must also ensure that reads and writes are not indefinitely stalled or starved while a stream of the opposite transaction is being performed on the bus. In particular, the system must ensure read/write fairness and avoid starvation for either.

[0007] The present invention is directed at an efficient system and architecture to maximize bandwidth and optimize system performance while avoiding bus contention and read/write starvation.

BRIEF SUMMARY OF THE INVENTION

[0008] In accordance with the present invention, there is provided a novel system and architecture for intelligently managing reads and writes on a bi-directional bus to optimize bus performance, avoid starvation, and introduce bubbles only when needed to avoid bus contention. This is accomplished by separately queuing reads and writes and then using a simple 2 to 1 multiplexer ("mux") to control the issuance of transactions and bubbles to the bus. The mux is controlled by bus streaming control logic which uses counters, comparators and certain threshold parameters to optimize the system performance by selectively streaming reads and writes together.

[0009] The inventive system for managing bi-directional bus usage, comprises: a bi-directional bus; a read queue for pending read bus transactions; a write queue for pending write bus transactions; a mux having a first input coupled to the read queue and a second input coupled to the write queue and an output coupled to the bus; bus streaming control logic coupled to the read queue and the write queue having a first and second control signal coupled to the mux; wherein the bus streaming control logic selectively controls the mux to issue or stream either read transactions from the read queue or write transactions from the write queue to the bus.

[0010] In an alternate embodiment of the present invention, the bus streaming control logic of the inventive system further comprises: a first counter to track the number of pending writes in the write queue; a second counter to track the number of consecutive reads issued to the bus; a third counter to track the number of pending reads in the read queue; and a fourth counter to track the

number of consecutive writes issued to the bus; as well as a first threshold for pending writes in the write queue; a second threshold for consecutive reads issued to the bus; a third threshold for pending reads in the read queue; and a fourth threshold for consecutive writes issued to the bus.

[0011] In an additional alternate embodiment of the present invention, the bus streaming control logic of the inventive system further comprises a plurality of counters to track the number of clock cycles that pending read transactions and write transactions have been waiting in the read queue and the write queue respectively and wherein the bus streaming control logic forces a transition to issue pending transactions which have been waiting for a predetermined amount of time in the queues.

[0012] The inventive computer system incorporating the present invention, comprises: a power supply; architecture for managing bi-directional bus usage, comprising: a read queue for pending read bus transactions; a write queue for pending write bus transactions; a mux having a first input coupled to the read queue and a second input coupled to the write queue; bus streaming control logic coupled to the read queue and the write queue having a control signal coupled to the mux; wherein the bus streaming control logic selectively controls the mux to output or stream read transactions from the read queue or write transactions from the write queue.

BRIEF DESCRIPTION OF THE DRAWINGS

[0013] The invention can be more fully understood by referencing the accompanying drawings wherein:

[0014] Fig. 1 shows a functional block diagram of the architecture 10 for managing bus transactions as contemplated by the present invention;

[0015] Fig. 2 illustrates read/write transactions issued on a bus prior to the present invention; and

[0016] Fig. 3 illustrates read/write transactions issued on a bus as contemplated by the present invention.

NOTATION AND NOMENCLATURE

[0017] Certain terms are used throughout the following description and claims to refer to particular system components. As one skilled in the art will appreciate,

components may be referred to by different names. This document does not intend to distinguish between components that differ in name but not function. In the following discussion and in the claims, the terms "including" and "comprising" are used in an open-ended fashion, and thus should be interpreted to mean "including, but not limited to... ." Also, the term "couple" or "couples" is intended to mean either an indirect or direct electrical connection. Thus, if a first device couples to a second device, that connection may be through a direct electrical connection, or through an indirect electrical connection via other devices and connections.

DETAILED DESCRIPTION OF THE DRAWINGS

[0018] Referring now to the drawings, wherein like reference characters denote corresponding components or parts:

[0019] Fig. 1 shows a functional block diagram of the system and architecture 10 for efficiently managing bus traffic as contemplated by the present invention. Reads and writes set for issuance to the bus are queued in a separate read queue 12 and write queue 14. These queues are collectively referred to herein as pending transaction queues 12, 14. Each queue 12, 14 is coupled to a multiplexer 16 or any other switch (collectively referred to herein as a "mux"). The output of the mux 16 is issued to the bus.

[0020] Bus streaming control logic 18 is coupled to the pending transaction queues 12, 14 to track or monitor the state of the read queue 12 and write queue 14. The bus streaming control logic 18 is coupled to the mux 16 via control signals Kill_Rd and Kill_Wr as shown. When the bus streaming control logic 18 asserts the Kill_Rd signal to the mux 16, all reads are blocked by the mux 16 and the mux 16 will only issue writes from the write queue 14 to the bus. When the bus streaming control logic 18 asserts the Kill_Wr signal, all writes are blocked by the mux 16 and the mux 16 will only issue reads from the read queue 12 to the bus. When both control signals are asserted, neither reads nor writes will be issued to the bus causing a bubble or delay on the bus. Of course if there are no pending transactions in queue, the bus will also be inactive. For example, if Kill_Rd is asserted and Kill_Wr is not asserted, there may still be a delay or bubble if there are no writes pending in the write queue 14. Similarly, if Kill_Wr is

asserted and Kill_Rd is not asserted, there may still be a delay or bubble if there are no reads pending in the read queue 12. The bus streaming control logic 18 utilizes counters, comparators and threshold parameters to selectively control the streaming of the reads, writes and bubbles to the bus. In various embodiments of the computer system incorporating the present invention, the bus streaming control logic 18 is in the Middle Mapper/Slotter of the Zbox as described in the applications incorporated herein by reference only.

[0021] The system and architecture 10 as contemplated by the present invention may be more fully understood by discussing an example of its operation. Reads and writes arrive at the pending transaction queues 12, 14 ready for issuance to the bus. Bus streaming control logic 18 uses a counter to track the number of pending reads in the read queue 12 (a "pending read counter") and a counter to track the number of pending writes in the write queue 14 (a "pending write counter"). When a stream of reads is being performed on the bus, the bus streaming control logic 18 compares the number of pending writes to a first threshold parameter. When the number of pending writes in the write queue 14 reaches the threshold number, the system is said to become "hungry" for writes and a transition from reads to writes is desired. Although the transition could be effectuated immediately by the bus streaming control logic 18, in various embodiments of the invention an additional criteria must be met before a transition is instituted. In particular, when the system becomes hungry for writes, the bus streaming control logic 18 starts another counter to track the number of consecutive reads issued (an "issued reads counter"). The bus streaming control logic 18 then compares the number of issued reads to a second threshold parameter. This second threshold parameter ensures that a sufficient stream of reads has been issued before a transition occurs. When the number of issued reads reaches the second threshold number, the system is said to be "starved" for writes and a transition from reads to writes is executed.

[0022] The control signals Kill_Rd and Kill_Wr are used to execute a transition from reads to writes. Since only reads are currently being issued to the bus, the Kill_Wr signal is already asserted to the mux 16. The bus streaming control logic 18 then asserts the Kill_Rd signal to stop all reads and writes from issuance to

the bus. With both control signals asserted, the mux 16 prevents any reads or writes from issuance to the bus and a bubble is implemented on the bus (*i.e.*, no reads or writes are issued). After a predetermined amount of time elapses, based on the necessary delay time required by the bus protocol, the bus streaming control logic 18 unasserts the Kill_Wr signal. With only the Kill_Rd signal asserted, the mux 16 begins to issue writes from the write queue 14 to the bus and the transition from reads to writes is complete.

[0023] Transitions from writes to reads are similarly handled. When a stream of writes is being performed on the bus, the bus streaming control logic 18 compares the number of pending reads to a third threshold parameter. When the number of pending reads in the read queue 14 reaches the threshold number, the system is said to become "hungry" for reads and a transition from writes to reads is desired. Although the transition could be effectuated immediately by the bus streaming control logic 18, in various embodiments of the invention an additional criteria must be met before a transition is instituted. In particular, when the system becomes hungry for reads, the bus streaming control logic 18 starts another counter to track the number of consecutive writes issued (an "issued writes counter"). The bus streaming control logic 18 then compares the number of issued writes to a fourth threshold parameter. This fourth threshold parameter ensures that a sufficient stream of writes has been issued before a transition occurs. When the number of issued writes reaches the fourth threshold number, the system is said to be "starved" for reads and transition from writes to reads is executed.

[0024] The control signals Kill_Rd and Kill_Wr are also used to execute a transition from reads to writes. Since only writes are being issued to the bus, the Kill_Rd signal is already asserted to the mux 16. The bus streaming control logic 18 then asserts the Kill_Wr signal to stop all writes and reads from issuance to the bus. With both control signals asserted, the mux 16 prevents any reads or writes from issuance to the bus and a bubble is implemented on the bus (*i.e.*, no reads or writes are issued). After a predetermined amount of time elapses, based on the necessary delay time required by the bus protocol, the bus streaming control logic 18 unasserts the Kill_Rd signal. With only the Kill_Wr signal

asserted, the mux 16 begins to issue reads from the read queue 12 to the bus and the transition from writes to reads is complete.

[0025] In order to accomplish this control of the bus usage, 4 counters and 4 threshold parameters have been used as follows:

Counters	Exemplary Thresholds
1. number of pending writes	1. threshold for pending writes (4)
2. number of consecutive reads	2. threshold for consecutive reads (8)
3. number of pending reads	3. threshold for pending reads (1)
4. number of consecutive writes	4. threshold for consecutive writes (8)

As discussed above, counters 1 and 3 continuously track and monitor the number of writes and reads pending in the write queue 14 and read queue 12, respectively. When counter 1 reaches the threshold 1 for pending writes, the system becomes hungry for writes and counter 2 is started to track the number of consecutive reads executed. When counter 2 reaches threshold 2 for consecutive reads issued, the system is starved for writes and a transition from reads to writes occurs. Similarly, when counter 3 reaches the threshold 3 for pending reads, the system becomes hungry for reads and counter 4 is started to track the number of consecutive writes executed. When counter 4 reaches threshold 4 for consecutive writes issued, the system is starved for reads and transition from writes to reads occurs.

[0026] The threshold parameters can be predetermined or programmable. Values for the parameters should be set to optimize system performance. Optimal values for the parameters will depend on the system. In various embodiments of the invention, the parameters are set as shown above. The threshold for pending reads is set to 1 and the threshold for pending writes is set to 4. This effectively gives a priority to reads since the system will become hungry for reads as soon as one read is pending in the read queue 12 (whereas four writes must be pending before the system becomes hungry for writes). These settings were used because in the computer system of at least some embodiments, system latency was more dependent on reads being delayed than writes. For any system where this is true, a similar preference for reads may be integrated in the threshold parameters. The thresholds for consecutive reads and

writes were both set to 8 in various embodiments because the read queue 12 and write queue 14 have 8 entries. Accordingly, when the system becomes hungry for a transition, one complete cycle through the queue is completed before the transition is allowed to take place. It is understood that one of the advantages of the invention is the ability to tune these parameters to maximize system performance. Accordingly, threshold parameters different than those used in the various embodiments may provide optimal performance in any given system.

[0027] It should be noted that in the discussion of the present invention to this point, there is the potential for a problem. In particular, when thresholds 1 or 3 are set to a number greater than 1, there is the potential for a read or write to become stranded in the pending transaction queues 12, 14 (*i.e.*, real or complete starvation of a read or write). For instance, if a stream of reads is being executed in various embodiments (where threshold 1 for pending writes is set to 4), when a write arrives at the write queue 14, it must wait until 4 writes have accumulated in the queue before the system will become hungry and begin counting the consecutive reads issued. If no other writes are forthcoming, the write could remain stranded in the write queue 14 indefinitely.

[0028] Obviously, indefinitely delaying a pending transaction could introduce additional and indefinite latency in the system as a whole. To avoid this possibility, an additional counter can be used to track the number of clock cycles a transaction has been waiting in the pending transaction queues 12, 14 and to ensure the transactions do not wait indefinitely for a transition. Specifically, in the various embodiments a counter is started when the first write enters the write queue 14. The counter is incremented each clock cycle to track the amount of time the write has been waiting. If a transition has not been completed in a predetermined amount of time, *i.e.*, before the counter reaches a predetermined value, then the system is forced to become hungry for writes (as if 4 writes had accumulated in the write queue 14). Accordingly, the counter 2 for consecutive reads is started and a transition occurs when it reaches the threshold 2 for consecutive reads. This assures that the write will not remain in the queue indefinitely waiting for 4 writes to accumulate. Such a counter is not required for reads in various embodiments because the threshold for reads is set to 1,

meaning the system becomes hungry for reads as soon as a read enters the read queue 12. Similar counters should be used if the thresholds 1 or 3 are set greater than 1 in order to ensure there is not indefinite starvation of reads or writes.

[0029] Fig. 2 and Fig. 3 attempt to illustrate the increased performance realized by the present invention. Fig. 2 illustrates reads and writes indiscriminately issued on a bus as would often occur prior to the present invention. In Fig. 2, reads are represented by an R on the bus and writes are represented by a W. Between each transition from read to write or vice versa, a bubble or delay is inserted to ensure there is no bus contention. Bubbles are represented by an X. In the example illustrated in Fig. 2, there are three reads and three writes being issued on the bus. Because the reads and writes are interspersed, however, several bubbles are required between the read/write transitions. Assuming each read, write and bubble takes a clock cycle, 11 cycles are required to issue the three reads and three writes as illustrated in Fig. 2.

[0030] Fig. 3 illustrates reads and writes issued on a bus as contemplated by the present invention. In particular, three reads and three writes are also depicted on the bus in Fig. 3. The reads and writes in Fig. 3, however, are streamed or grouped together as opposed to intermingled as shown in Fig. 2. The result is that only one bubble is required since there is only one transition between reads and writes. Accordingly, only 7 cycles are required to issue three reads and three writes (again assuming each read, write and bubble takes one clock cycle). Thus, by grouping or streaming the reads and writes, 4 cycles were saved from the example in Fig. 2. The invention streams the reads and writes to allow more transactions to occur on the bus with less bubbles or latency. The result is increased system performance.

[0031] The above discussion is meant to be illustrative of the principles and various embodiments of the present invention. While the invention has been particularly shown and described with respect to specific embodiments thereof, numerous variations and modifications will become apparent to those skilled in the art once the above disclosure is fully appreciated. It is intended that the following claims be interpreted to embrace all such variations and modifications.